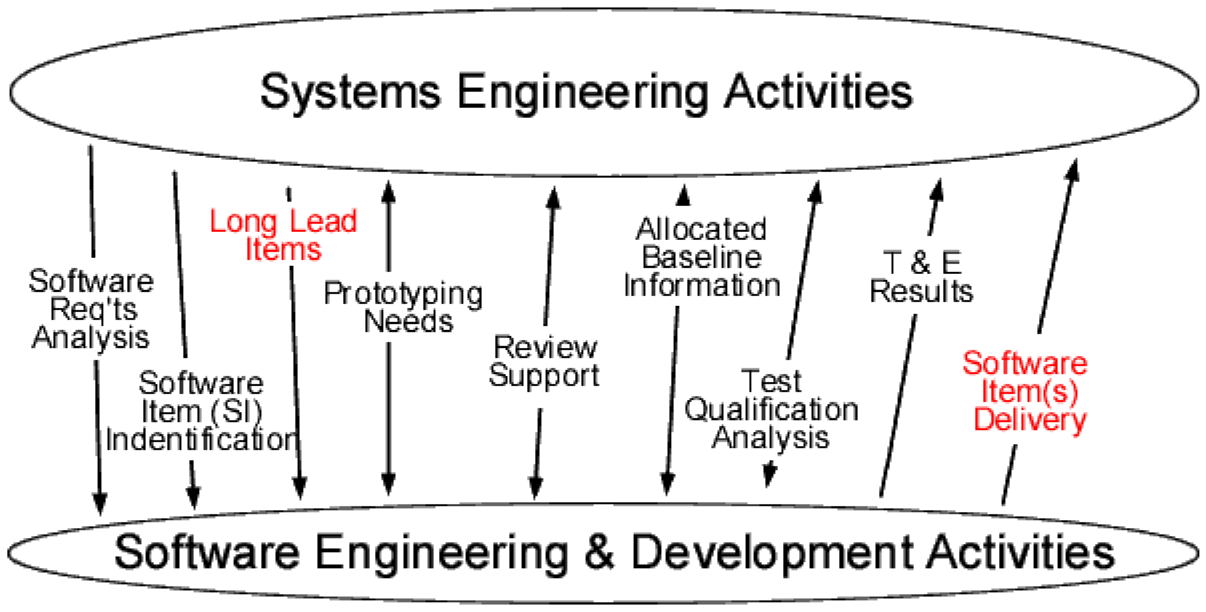
REVIEW RESOURCES

Lesson 23: Software Acquisition: Development and Management

The Linkage Between Systems Engineering and Software Development

The DOD policy is to design and develop software based on Systems Engineering principles. Some examples of the linkages based on these principles are shown below.


Systems Engineering Activities
Software Req'ts Analysis
Software Item (SI) Indentification
Long Lead Items
Prototyping Needs
Review Support
Allocated Baseline Information
Test Qualification Analysis
T & E Results
Software Item(s) Delivery
Software Engineering & Development Activities

Back to Topics List Back to Top

Why Worry About Software?

Software is an integral part of most defense systems. Following are two reasons why software development requires special attention:

- Software development can be difficult to monitor because it is a complex, changeable, and invisible product.
- Once software development is late, adding more resources typically results in making the project even later.

Back to Topics List Back to Top

Software Risk Management

Most software development projects are very large and complex. Breaking a software-intensive project into more manageable parts helps the people involved better understand the tasks and resources needed. By understanding the complexity of an effort, managers are better able to plan and manage the risk.

Software Items (SIs): The Building Blocks

As part of the Systems Engineering Process, the software is usually broken down into smaller building blocks called Software Items (SIs). After being developed and individually tested, Software Items are integrated with the hardware and ultimately the entire system.

A Software Item (SI) is a collection of software that performs closely related functions. Each SI is uniquely designated throughout the life cycle for purposes of:

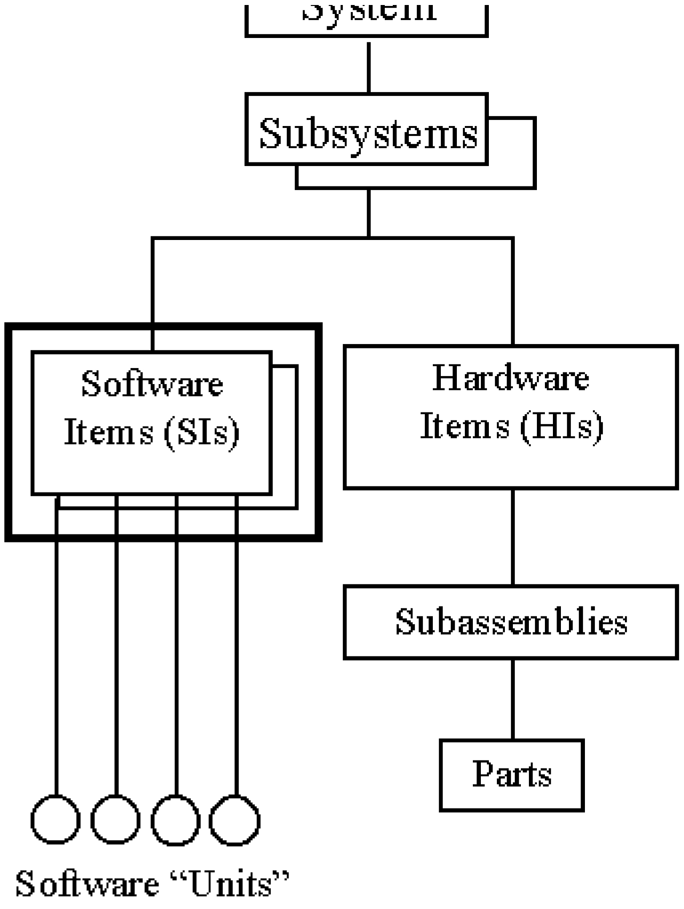
- Managing requirements.
- Conducting qualification testing.
- Controlling interfaces.
- Ensuring configuration management.
- Mitigating risk.

Note: In the past, Software Items were called Computer Software Configuration Items (CSCI).

SIs and Work Breakdown Structures

As you learned in a previous lesson, a Work Breakdown Structure (WBS) divides complex projects into pieces so that risks can be identified and managed.

As illustrated below, WBS can be used to organize the Software Items (SIs) and to show the relationship among the different system components.



Software Units

To facilitate programming, Software Items (SIs) can be broken down into smaller, logically related pieces. Many commercial standards call these pieces "Software Units."

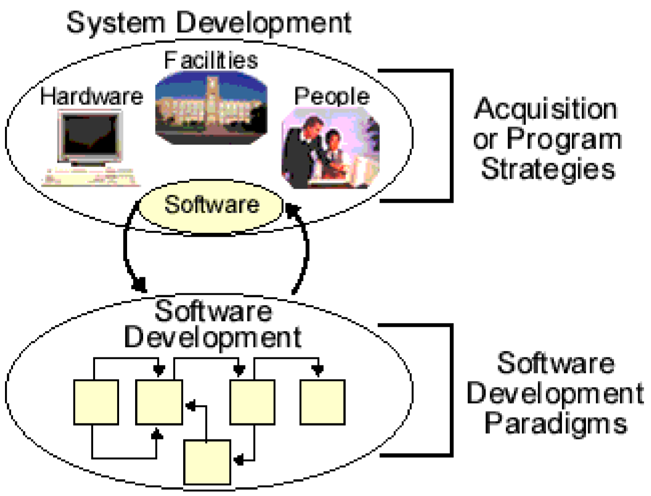
Software Units include individual programs such as modules or routines that perform specific functions.

Back to Topics List Back to Top

Development Paradigms

In an earlier lesson, you learned about the importance of structuring a tailored acquisition or program strategy.

The software developer uses a similar structured approach to lay out software development tasks. These different ways of laying out tasks are referred to as "software development paradigms."



What Are Paradigms?

In a software engineering context, the approaches to structuring software development are called "Software Development Paradigms." A more common word for paradigms is models.

Types of Paradigms (Models)

The three common software development models are called:

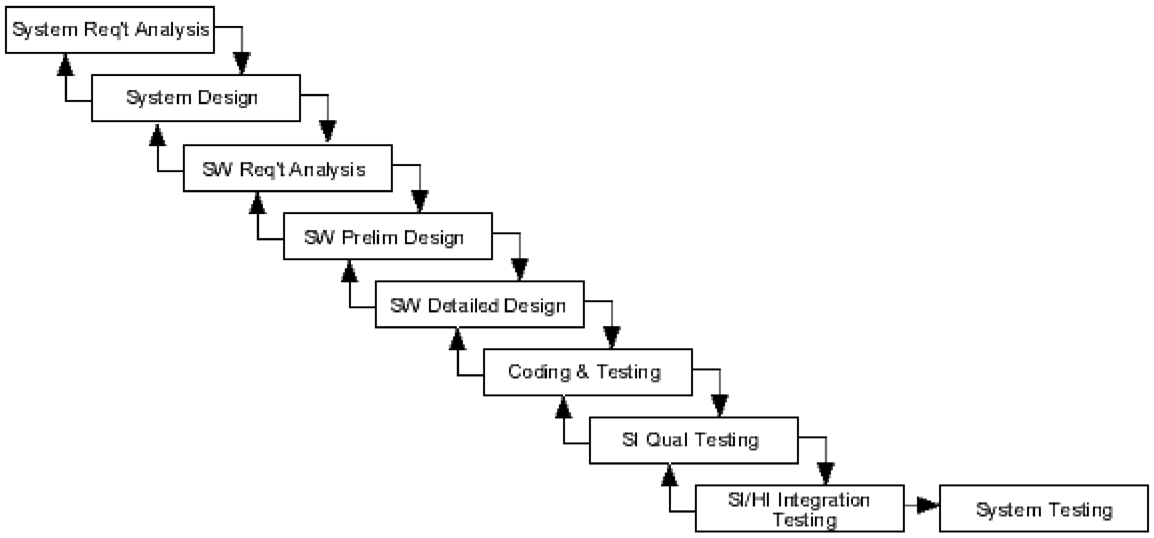
- Waterfall Paradigm
- Incremental Paradigm
- Spiral Model

Waterfall Paradigm

The Waterfall Paradigm is implemented using a sequential process. In the traditional Waterfall Paradigm:

- Each stage is a prerequisite for succeeding activities.
- Successful completion of a stage is required before starting the next one.
- Formal reviews are used to assess the completion of each stage.

As illustrated below, the developer must be able to accurately estimate the difficulty of all required steps. Therefore, the Waterfall Paradigm works best with a precedented software-intensive system.



Precedented System

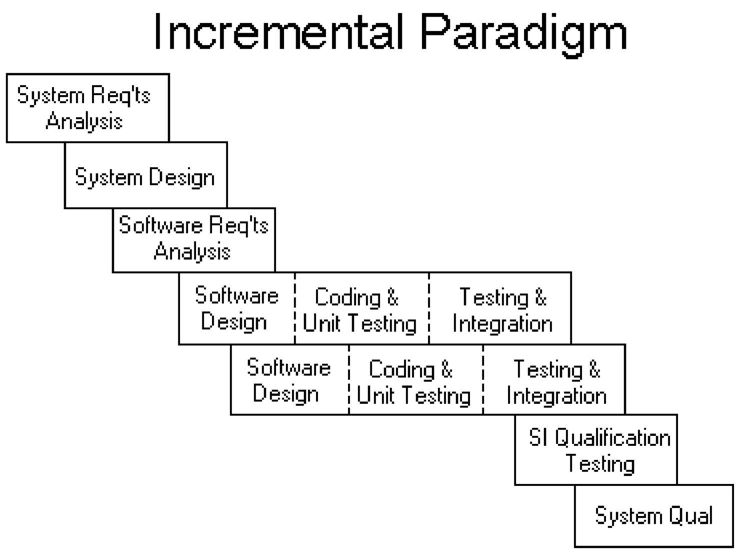
A Precedented System includes the following characteristics:

- Stable system and software requirements, not significantly different than those of previously developed system(s).
- Systems engineering and software development teams with previous experience developing similar systems.

Incremental Paradigm

The Incremental Paradigm involves developing "pieces" of the software for a system in a series of incremental steps. There are many variations of this paradigm. When an incremental approach is used, it is important to make sure that the developer is prepared to build and integrate all of the increments. When using this model, special emphasis should be placed on the selection of the items to be included in the last increment. Sometimes the last increment can be the hardest to complete!

When incremental development is used, an Integrated Product Team (IPT) may be used to help define the number, size, and phasing of the increments. Involving the IPT can reduce risk and ensure that all of the user's requirement will be satisfied by the increments.



Spiral Model

The Spiral Model is a risk-driven prototyping approach to software development. This model:

- Identifies high-risk areas (e.g., user interface, cost, performance constraints, evolving requirements, etc.) and develops software prototypes starting with the highest risk first.
- Allows development to begin before the final design is fully evolved.
- Uses feedback from these prototypes for mitigating risks.

Spiral Model development is especially appropriate for unprecedented software-intensive systems.

Key characteristics of the Spiral Model include:

- Risk based-prototyping.
- Feedback from prototype.
- Elaboration of draft documentation during the life cycle.
- Repetition until risk is at an acceptable level.

Unprecedented System

An Unprecedented System is one that has not been developed before. No models of previous system (s) or experience base exist on which to base the development of a new, similar system.

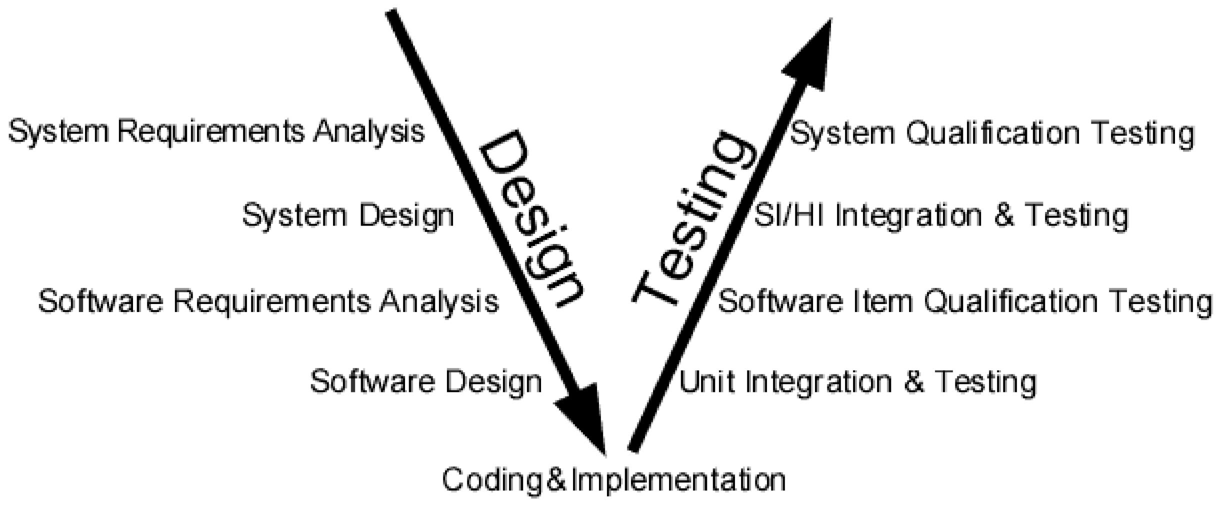
Back to Topics List Back to Top

Software Development

This section presents key activities related to software development. The use of a software development paradigm does not negate the need to implement an overall systematic development process within that paradigm.

Overview of Software Development

Typically, software is designed using a "top-down" strategy. After software coding and implementation, testing and integration follow a "bottom-up" strategy. When illustrated, the software development process looks like a V-shape.


System Requirements Analysis
System Design
Software Requirements Analysis
Software Design
Design
Coding & Implementation
Testing
Unit Integration & Testing
Software Item Qualification Testing
SI/HI Integration & Testing
System Qualification Testing

System Requirements Analysis and System Design

During these steps, the Systems Engineering Process:

- Develops complete and consistent top-level specifications.
- Determines which requirements are best performed through software.

Software Requirements Analysis

Based on the overall Systems Design, the software developer then:

- Helps to finalize the selection of Software Items (SIs).
- Determines detailed requirements for each SI.

Software Requirements Specification

One key to effective software development is well defined software requirements. The result of the Software Requirements Analysis step is a detailed software requirements specification. Software requirements are typically detailed in a Software Requirements Specification (SRS). This document becomes the "blueprint" for each Software Item to be developed.

Unstable Requirements

Software requirements specifications are important. Well designed and defined requirements can prevent "requirements creep" that can lead to unstable requirements. Unstable and ill-defined requirements are major cost and schedule drivers.

Building Stable Requirements

To ensure stability, requirements should be stated so that they are:

- **Traceable:** Requirements should be able to be tied to system and user requirements.
- **Understandable:** Users and developers should be able to understand the requirements.
- **Achievable:** The developer should be able to achieve the requirements within the estimated

cost.

- **Measurable:** Developers and users should be able to measure the requirements during testing.

Software Design

After the requirements are stabilized, the developer builds detailed designs for each Software Item. At this point it is critical to ensure that all requirements are addressed. Requirements are carefully traced from one development stage to the next.

Coding and Implementation

During this phase the developer writes or implements the code for each Software Unit. Each Software Unit is then tested in a stand-alone mode. Remember, each Software Item can be arranged into several Software Units.

Comprehensive Testing

The focus of the development process now shifts to conducting comprehensive computer-based testing. Comprehensive testing demonstrates that the delivered software will satisfy the requirements. Although not mentioned before, some human-based software testing begins much earlier.

Types of Software Testing

Software testing can be divided into the broad categories of human-based and computer-based testing. The type of testing performed depends on the software development stage.

Human-Based Testing

Human-based testing is a method of examining:

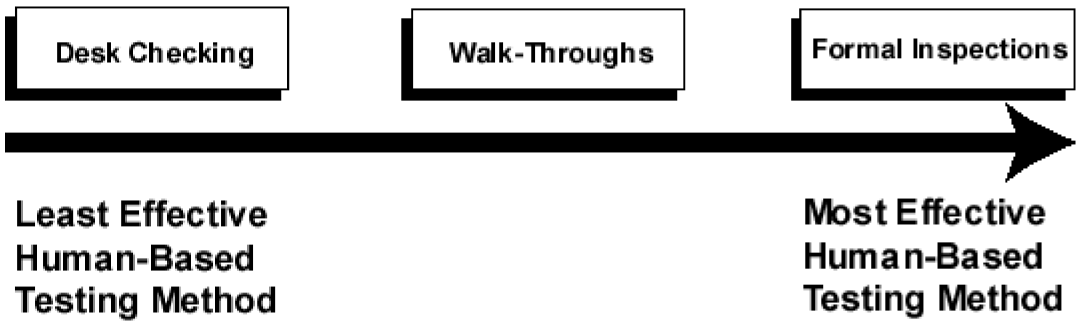
- Requirements
- Design
- Code

This type of testing is primarily a manual process. Testing begins as early as possible in the software development life cycle.

Human-based testing identifies software errors early when those errors are cheapest to correct. This type of testing is critical for ensuring software quality.

Examples of Human-Based Testing

Examples of human-based testing include:


Desk Checking
Walk-Throughs
Formal Inspections
Least Effective Human-Based Testing Method
Most Effective Human-Based Testing Method

Desk Checking

Desk checking involves individuals reviewing the quality and correctness of their own work. Although commonly used, desk checking is not very effective at uncovering errors.

Walk-Throughs

During a walk-through, the author of the product being evaluated makes a presentation of the product (e.g., software requirements, design, and code) to a group for review.

The presentation is followed by a general discussion from the participants after which the presenter receives feedback on the specific item being reviewed.

Formal Inspections

Formal Inspections (also called Peer Reviews):

- Are led by a trained, impartial moderator.
- Involve inspection teams that use established criteria to review the software product (e.g., software requirements, design, and code).
- Are done primarily by the software developer. However, the Government may participate at requirement-level reviews.
- Are highly effective when done properly.

Computer-Based Testing

Computer-based testing:

- Refers to those testing activites that start after coding is complete.
- May use a variety of Computer-Aided Software Engineering (CASE) tools.

Computer-Based Testing: Examples

Examples of Computer-Based Testing include:

- Software Unit Testing
- Software Unit Integration and Testing
- Software Item Qualification Testing
- Hardware Item and Software Item Integration and Testing
- Subsystem Integration and Testing
- System Qualification Testing

Software Unit Testing

Software Unit Testing is a low-level test of an individual Software Unit. Software units are typically routines, modules, or smaller portions of a larger program.

Software Unit Integration and Testing

Many Software Units make up a Software Item. Unit Integration and Testing involves progressive integration of the units making up an SI and testing their operation with one another, including internal interfaces.

Software Item Qualification Testing

After all the units have been integrated and tested, the Software Item they comprise is tested as an entity. The basis of this test, which can be called "Qualification Testing," is typically described in the Software Requirements Specification (SRS).

Hardware Item and Software Item Integration and Testing

This level of testing involves integration and testing of the various Hardware Items (HIs) and Software Items (SIs) that can make up a subsystem.

Subsystem Integration and Testing

This test involves the integration, testing, and qualification of the various subsystems that make up the complete system being developed.

System Qualification Testing

This type of testing is a high-level technical test designed to qualify the entire system against its systems-level requirements, which are typically described in the System Specification.

Managing Development Risk

Development of some categories of software-intensive systems, especially embedded systems, may be high risk. Some examples of these high risk categories include:

- Real-time critical software systems that generate some action in response to external events under severe time and reliability constraints.
- Programs having a high cost of failure in terms of human life, national security, or mission success.

For these inherently risky systems, the Program Manager may choose to use an appropriate level of Independent Validation and Verification (IV&V).

Independent Verification and Validation

Independent Verification and Validation (IV&V) is the systematic evaluation of software products and activities by an agency that is not responsible for the system development.

Verification involves confirming that either a result is correct or that a procedure or sequence of operations has been performed. An example of Verification is determining that the software requirements analysis is complete and that software design activities can be started.

Validation involves testing to ensure that software requirements are implemented correctly and completely. An example of Validation is acceptance testing, where the system is put through its paces and its performance is validated against the requirements.

Metrics

In order to determine the status of an acquisition, the Program Manager needs to measure the development progress achieved. DOD policy mandates the use of a software measurement program. Proper use of software measurement (also called metrics or indicators) can provide early insight into cost and schedule risks and product quality.

A software metric is any measure that will allow a manager to measure and evaluate the products and/or the processes of a software development project.

Software Metrics Categories

Software metrics can be classified into the following three categories:

- Management
- Quality
- Process

The choice of the particular metrics to use on a given program should be driven by risk. Programs should measure those areas having the greatest risk impact.

Management Metrics

Management metrics are:

- Primarily concerned with indicators that help determine progress against plan.
- Measures selected from indicators that have an impact on the effect, cost, and schedule of the required effort.

Examples of management metrics include:

- Software Size
- Software Personnel Status
- Computer Resource Margins
- Software Requirements Volatility
- Development Progress
- Test Program Status
- Status of Software Problem Reports

Quality Metrics

Quality metrics help assess product quality attributes such as:

- Performance
- Ease of change
- User satisfaction
- Supportability

Examples of quality metrics include:

- Error Density
- Reliability
- Usability
- Complexity
- Portability
- Correctness
- Modularity
- Maintainability

Process Metrics

Process metrics are indicators that deal with the maturity and robustness of the processes involving organizations, tools, techniques, and procedures that are used to develop and deliver software products.

Organizations with a proven track record with processes, tools, techniques, and procedures in place are considered mature.

Following are examples of models that can be used to assess the maturity of an organization's software development process:

- Software Capability Evaluation Models based on the Software Engineering Institute's (SEI's) Software Capability Maturity Model (SW-CMM).
- U.S. Air Force’s Software Development Capability Evaluation (SDCE) Model.

Benefits of Software Measurement Programs

When used properly, a software measurement program can:

- Provide Program Managers with information they need to make any needed adjustments to balance cost, schedule, and performance objectives.
- Help ensure the delivery of a quality product that will satisfy users’ needs.
- Generate products that may be more easily and economically supported.

Post Deployment Software Support (PDSS)

When a software-intensive system is fielded, it is only the beginning of an on-going process to maintain, update, and support the software. Within the DOD, this on-going process is frequently referred to as "Post Deployment Software Support (PDSS)" or "Post Production Software Support (PPSS)." Software support is also referred to as "Software Maintenance."

Purpose

The purpose of software support is to ensure that, during the operational life of a software-intensive system, the software continues to support its operational mission. In addition, the software must support any subsequent mission modifications.

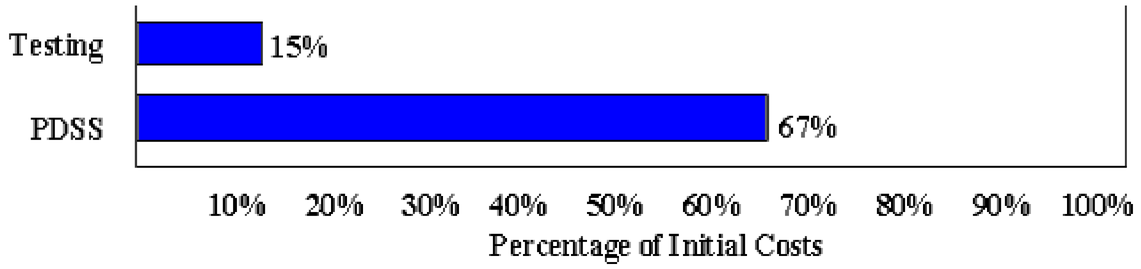
Software Support Activities

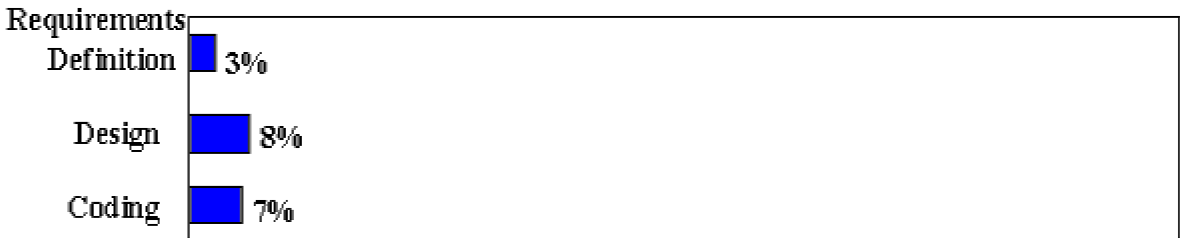
The activities for ensuring that fielded software continues to fully support the operational mission of the system include:

- Correcting problems.
- Providing new functions.
- Adapting the product to a modified environment.

Software Life Cycle Cost Percentages

As illustrated below, Requirements Definition, Design, Coding, and Testing make up only 33 percent of the software development life cycle costs. At 67 percent, PDSS is the greatest portion of the software system life cycle costs.





Best Practices

Successful Program Managers have learned the importance of early planning for support and maintenance activities. The following lessons have been captured as "best practices."

- Plan early! Identify anticipated areas of software changes and enhancements.
- Address cost, performance, schedule, and PDSS.
- Establish software support concepts and acquire necessary PDSS resources.
- Address software maintenance concepts for systems that include commercial-off-the-shelf (COTS) products.
- Include PDSS in planning documentation.

Back to Topics List Back to Top